

Transição para C#

Para programadores de JAVA

Programas simples	JAVA	C#		
	<pre>// Program.java package MyProgram; import java.lang.System; /** * * @author Jose Cordeiro */ public class Program { public static void main(String[] args) { System.out.println("Hello World!"); } }</pre>	<pre>// Program.cs using System; namespace MyProgram { /// <summary> /// ola mundo /// </summary> public class Program { public static void Main(string[] args) { Console.WriteLine("Hello World!"); } } }</pre>		
	Em C# usam-se <i>namespaces</i> com uma função equivalente aos <i>packages</i> do JAVA, agrupar classes relacionadas. A diretiva using do C# é semelhante ao import do JAVA. Permite usar tipos de outro <i>namespace</i> sem ter de os especificar. Em C# os comentários em XML, equivalentes ao Javadoc do Java, iniciam-se por /// e originam ficheiros em XML. O método main do JAVA chama-se Main em C# seguindo a convenção de que os identificadores públicos começam por maiúsculas (em notação Pascal Case)			
Tipos de dados	JAVA	C#		
	Primitive types byte ---- short ---- int ---- long ---- char float double boolean ----	Built-in types sbyte byte short ushort int uint long ulong char float double bool decimal	Size 8-bit 8-bit 16-bit 16-bit 32-bit 32-bit 64-bit 64-bit 16-bit 32-bit 64-bit true ou false Até 28 casas decimais	Struct SByte Byte Int16 UInt16 Int32 UInt32 Int64 UInt64 Char Single Double Boolean Decimal
	Em C#, os tipos prédefinidos são nomes alternativos (<i>alias</i>) para as estruturas associadas a esses tipos. Estes tipos são <i>value types</i> significando que é guardado diretamente o seu valor (e não a referência desse valor).			
Tabelas	JAVA	C#		
	String parameters[] String[] parameters	---- string[] parameters		
	Em C# os parenteses retos usados na declaração de <i>arrays</i> aparecem sempre a seguir ao nome do tipo			
	JAVA	C#		
	int[][] values = new int[15][]; for(int i=0; i<15; i++) values[i] = new int[10];	int[,] values = new int[10,15];		
Em C# existem <i>arrays</i> multidimensionais, ao contrário do java, que apenas tem tabelas de tabelas (também possíveis em C#). Na criação destes <i>arrays</i> multidimensionais, todas as células são criadas automaticamente.				

Seleção	JAVA	C#
	<pre>switch(j) { case 1: x = "one"; break; case 2: x = "two"; case 3: x = "three"; break; default: x = "NA"; break; }</pre>	<pre>switch(j) { case 1: x = "one"; break; case 2: x = "two"; goto case 3; break; case 3: x = "three"; break; default: x = "NA"; break; }</pre>
<p>Na instrução switch do C# o break é obrigatório para todos os case. Pode usar-se uma instrução "goto case 1", para reproduzir o que se faz em JAVA quando se omite o break</p>		
Repetição	JAVA	C#
	----	Label: <instruction>; goto Label;
	<p>O C# permite a utilização de uma instrução goto <label> para alterar a sequência de instruções embora não se recomende o seu uso. Neste caso uma utilização aceitável é para a saída de ciclos encadeados.</p>	
	JAVA	C#
	<pre>for(int val : numbers) System.out.print(val);</pre>	<pre>foreach(char c in str) Console.Write(c);</pre>
<p>O ciclo foreach em C# pode ser usado com qualquer classe que implemente a interface IEnumerable, incluindo <i>strings</i></p>		
Operadores e Instruções	JAVA	C#
		<pre>using (type var = value) { ... }</pre>
	<p>A instrução using permite usar a memória local para uma determinada variável que é válida dentro do bloco do <i>using</i>. A classe dessa variável deverá implementar a interface IDisposable onde está declarado o método Dispose(). Este método é sempre chamado (implicitamente) no fim do bloco.</p>	
	JAVA	C#
	<pre>instanceof ----</pre>	<pre>is as</pre>
	<p>O operador as em C# permite fazer um cast de um objeto para um determinado tipo sem que seja gerada uma exceção se os tipos forem incompatíveis. Neste caso é devolvido o valor null. O operador is funciona como o instanceof do Java.</p>	
	JAVA	C#
	----	checked e unchecked
	<p>checked e unchecked podem ser usados para controlar um pedaço de código (entre parênteses ou dentro de um bloco a seguir a estas instruções) gerando ou não exceções, respetivamente, no caso de <i>overflows</i> em expressões aritméticas.</p>	
	JAVA	C#
	<pre>string s1, s2; if(s1.equals(s2)) { }</pre>	<pre>string s1, s2; if(s1 == s2) { }</pre>
<p>A comparação dos textos guardados em Strings em C# é feita usando o operador ==. Para comparar referências deve-se usar o método ReferenceEquals (equivalente ao funcionamento do == do Java).</p>		

Redefinição de operadores	JAVA	C#
	----	<pre> public class Complex{ private double im, re; public Complex(double real, double imaginary) { im = imaginary; re = real; } public static Complex operator + (Complex a, Complex b) { return new Complex(a.re+b.re, a.im+b.im); } } </pre>
	<p>Em C#, é possível redefinir operadores de uma forma semelhante ao que se faz em C++.</p> <p>Os métodos de redefinição dos operadores devem ser sempre public e static e levar como argumentos os operandos.</p> <p>Nem todos os operadores do C# podem ser redefinidos.</p>	
	JAVA	C#
		<pre> public class AccountBalance { private int balance; public static implicit operator int(AccountBalance a) { return balance; } public static explicit operator string(AccountBalance a) { return "\$" + balance; } } AccountBalance account = new AccountBalance(); int balance = account; // implicit conversion string str = string(account); // explicit conversion </pre>
	<p>Em C#, tal como em C++, é possível redefinir os operadores de cast usados para converter valores de um tipo noutra tipo. Neste caso, redefinem-se de uma forma semelhante aos outros operadores aparecendo o nome do tipo no lugar do operador. As conversões definidas deste modo pelo utilizador podem ou não vir a necessitar de explicitar o casting através da aplicação, respetivamente, da palavra chave explicit ou implicit</p>	
Métodos	JAVA	C#
	----	<pre> public static void Add(ref int a, int b) { a += b; } ... ClassName.Add(ref a, b); ... </pre>
	<p>Em C# os <i>reference types</i> (todos os objetos) são passados por valor (tal como os <i>value type</i>). Neste caso, o que é passado é o "valor da referência". No entanto também é possível passar um <i>value type</i> ou <i>reference type</i> por referência desde que se assinala o facto com a palavra-chave ref antes do parâmetro na <i>declaração do método</i> e antes do valor passado como argumento na <i>chamada do método</i>. No caso do método ser usado para receber o valor que irá ser colocado na variável passada substitui-se a palavra ref pela palavra out tanto na assinatura do método como na chamada do mesmo, evitando-se assim ter de inicializar essa variável antes da chamada ao método. Também se pode usar o modificador in para um parâmetro de um método. Neste caso, significa que o parâmetro é passado por referência mas que não poderá ser modificado dentro do método, é apenas de leitura.</p>	

Número variável de parâmetros	<div> <div>JAVA</div> <pre> public class Drawing { // ... public void add(Figure... figs) { for(Figure fig : figs) { figures.add(fig); } } } // Example Drawing draw = new Drawing(); draw.add(new Circle()); draw.add(new Circle(), new Square()); </pre> </div> <div> <div>C#</div> <pre> public class Drawing { // ... public void Add(params Figure[] figs) { foreach (Figure fig in figs) { figures.Add(fig); } } } // Example Drawing draw = new Drawing(); draw.Add(new Circle()); draw.Add(new Circle(), new Square()); </pre> </div>	
	<p>Quando se pretende passar para um método um número variável de valores do mesmo tipo é habitual usar um <i>array</i>. Para simplificar este processo, em C#, utiliza-se a palavra-chave param, antes do parâmetro que recebe os valores, para indicar que esse parâmetro recebe um número variável de valores que serão colocados no <i>array</i>. Depois basta passar o número de valores que se quiserem, separados por vírgulas, no lugar do argumento. É equivalente ao <i>varargs</i> do Java que utiliza as reticências (...) antes do parâmetro com o mesmo objetivo. Em C#, o parâmetro é obrigatoriamente um <i>array</i> unidimensional e, tal como em Java, deve ser o último parâmetro do método.</p>	
Parâmetros opcionais	<div> <div>JAVA</div> <pre> ----- </pre> </div> <div> <div>C#</div> <pre> public class Example { public void ExampleMethod(int required, string optionalstr = "default", int optionalint = 10) { // ... } } // Call: Example ex = new Example(); ex.ExampleMethod(10); // optionalstr="default", optionalint=10 ex.ExampleMethod(17, "Other"); // optionalint=10 ex.ExampleMethod(25, "Other", 12); </pre> </div>	
	<p>Em C#, na definição de um método, é possível ter parâmetros opcionais. Para isso é apenas necessário fornecer o valor de omissão para esse parâmetro que é o valor com que fica se o valor não for fornecido. Os parâmetros opcionais devem ser os últimos do método, não podendo existir nenhum parâmetro não opcional entre os parâmetros opcionais.</p>	
Parâmetros nomeados	<div> <div>JAVA</div> <pre> </pre> </div> <div> <div>C#</div> <pre> public class Order { public void PrintDetails(string seller, int order, string product) { // ... } } PrintDetails("Gift Shop", 31, "Red Mug"); PrintDetails(order:31, product:"Red Mug", seller: "Gift Shop"); PrintDetails("Gift Shop", 31, product: "Red Mug"); </pre> </div>	
	<p>Em C# é possível fornecer os argumentos de um método em qualquer ordem desde que sejam precedidos pelo nome do parâmetro seguido de dois pontos (:). No entanto, se se pretender usar também algum dos argumentos normais sem o nome, então deve-se obedecer à ordem em que aparecem na definição do método.</p>	

Estruturas	JAVA	C#
	----	<pre> public struct Point { public int x; public int y; public Point(int x, int y) { this.x = x; this.y = y; } public void Print() { Console.WriteLine("x = {0}, y = {1}", x, y); } } </pre>
	<p>À semelhança do C++, em C# existe um tipo de dados semelhantes às classes denominado struct. Este tipo de dados apresenta as seguintes diferenças em relação às classes:</p> <ul style="list-style-type: none"> - A herança não se aplica a estruturas, mas podem implementar interfaces. - Os dados deste tipo são <i>value types</i> sendo, como tal, guardados e passados por valor. - As estruturas não têm um construtor por omissão. 	
Tipos enumerados	JAVA	C#
		<pre> enum Gender : byte { Male, Female } </pre>
	<p>Em C# é possível usar tipos enumerados tal como em Java. Os tipos enumerados normalmente usam o tipo int para guardar o seu valor. Em C# pode-se especificar outro tipo inteiro qualquer (por exemplo byte). Os valores de um tipo enumerado usam a convenção PascalCase. Em C#, não é possível redefinir o método <i>toString</i> dentro do tipo enumerado tal como é feito em Java. Uma possível solução para este problema passa por criar um <i>método de extensão</i> para o tipo enumerado com esse objetivo.</p>	
Finalizar classes e variáveis	JAVA	C#
	<pre> public final class FinalCircle { public static final double Radius = 1.5; private final int id = 1; } </pre>	<pre> public sealed class FinalCircle { public const double radius = 1.5; private readonly int id = 1; } </pre>
	<p>Em Java, a palavra final tem diferentes utilizações que em C# são diferenciadas usando-se palavras-chave diferentes. Assim, sealed é usada para <i>selar</i> uma classe não permitindo a derivação de outras classes, readonly é usado para declarar uma variável apenas de leitura e const para declarar constantes. Neste último caso as constantes são por omissão variáveis estáticas apenas de leitura.</p>	

Herança e Polimorfismo	JAVA	C#
	<pre> public class Figure { private int x,y; public Figure() { this(0, 0); } public Figure(int x, int y) { this.x = x; this.y = y; } public String getValues() { return "(" + x + "," + y + ")"; } } // End Class Figure public class Circle extends Figure { private int radius = 1; public Circle(int x, int y, int radius) { super(x, y); this.radius = radius; } public String getValues() { return super.getValues() + "r=" + radius; } } // End Circle </pre>	<pre> public class Figure { private int x,y; public Figure() : this(0, 0) { } public Figure(int x, int y) { this.x = x; this.y = y; } public virtual string GetValues() { return "(" + x + "," + y + ")"; } } // End Class Figure public class Circle : Figure { private int radius = 1; public Circle(int x, int y, int radius) : base(x, y) { this.radius = radius; } public override string GetValues() { return base.GetValues() + "r=" + radius; } } // End Circle </pre>
	<p>A herança em C# é feita usando o símbolo : em vez do extends do Java.</p> <p>Em C#, a chamada ao construtor da classe base é feita a seguir à declaração do construtor e antes do bloco de código do mesmo. Neste caso, acrescenta-se dois pontos, a palavra base e os parênteses com o(s) argumento(s) a passar ao construtor da classe base. Esta palavra-chave corresponde ao super do Java.</p> <p>Ao contrário do Java, os métodos em C# não são virtuais sendo necessário declará-los como tal usando a palavra-chave virtual. Os métodos abstratos são virtuais por defeito.</p> <p>Para redefinir um método virtual numa classe derivada em C# é necessário acrescentar a palavra-chave override. A alternativa é usar a palavra new significando de que se trata de um novo método e não de uma redefinição.</p>	
Interfaces	JAVA	C#
	<pre> public interface Drawable { public void draw(); } public class Person implements Drawable { ... } </pre>	<pre> public interface IDrawable { public void Draw(); } public class Person : IDrawable { ... } </pre>
	<p>Em C#, os identificadores das interfaces começam, por convenção, pela letra I maiúscula. Uma classe que implemente uma ou mais interfaces coloca o(s) seu(s) identificador(es) depois do nome da classe e a seguir ao símbolo : tal como para a herança. Sendo assim, a seguir a ":" vem o nome da classe base seguido dos identificadores das interfaces que a classe implementa, todos separados por vírgulas. Ao contrário do Java, que usa a palavra extends antes do nome da classe base e a palavra implements antes do nome das interfaces, em C# não há distinção desses dois tipos, daí a utilização do prefixo I no nome das interfaces para as distinguir da classe.</p>	
Exceções	JAVA	C#
	Exception and error java.lang.Throwable	Exception System.Exception
	<p>O C# têm uma hierarquia de exceções tal como em Java só que neste caso a classe base é System.Exception. Também em C# não são feitas distinções entre exceções e erros.</p> <p>A grande diferença em relação ao Java, é que não é necessário, nem possível, declarar a seguir à assinatura do método as exceções que são lançadas dentro desse método.</p> <p>Existe também em C# uma forma especial de criar um bloco catch em que não aparecem os parênteses com a exceção gerada. Neste caso qualquer exceção que apareça leva à execução do bloco desse catch.</p>	

Propriedades	JAVA	C#
	<pre>public class Circle { private int radius; public void setRadius(int value) { this.radius = value; } public int getRadius() { return radius; } }</pre> <p>Utilização:</p> <pre>Circle c = new Circle(); c.setRadius(10); System.out.println("Radius=" + c.getRadius());</pre>	<pre>public class Circle { private int radius; public int Radius { set { radius = value; } get { return radius; } } }</pre> <p>Utilização:</p> <pre>Circle c = new Circle(); c.Radius = 10; Console.WriteLine("Radius=" + c.Radius);</pre>
	<p>Em C# é introduzido o conceito de propriedade usado nos componentes. Uma propriedade é usada como se fosse um atributo público mas o acesso é na realidade feito através de métodos get e set. Para se ter acesso ao valor recebido por uma propriedade dentro da sua classe usa-se a palavra-chave value como se fosse uma variável. Esta <i>variável</i> contém o valor recebido.</p> <p>É possível omitir o set ou o get de uma propriedade tornando-a assim, respetivamente, só de leitura ou só de escrita.</p> <p>Uma propriedade pode ser declarada como abstrata ou como virtual e pode ser usada em interfaces.</p>	
Indexers	JAVA	C#
	<pre>----</pre>	<pre>public class Screen { private char[,] screen = new char[25, 80]; public char this[int x, int y] { get { return array[x,y]; } set { ecra[x,y] = value;} } }</pre> <pre>Screen screen = new Screen(); screen[2,3] = 'c'; Console.Write(screen[2, 3]);</pre>
	<p>Em C# é possível ter a notação usada com <i>arrays</i> em classes normais através de <i>indexers</i>. Neste caso definem-se os <i>indexers</i> de uma forma semelhante às propriedades com um método set usado para alterar o elemento do <i>array</i> e um método get usado para receber um elemento do <i>array</i>.</p> <p>O parâmetro usado no índice dos <i>indexers</i> pode ser de qualquer tipo e pode, inclusive, fornecer-se mais que um parâmetro como é feito em <i>arrays</i> multidimensionais. No exemplo mostra-se uma classe Screen com um <i>indexer</i> contendo 2 índices correspondendo a coordenadas dentro do ecrã.</p> <p>Tal como nas propriedades é possível omitir o bloco get ou o bloco set.</p>	

Delegates e Eventos	JAVA	C#
		<pre> public class Stock { ... public void ProcessPriceChanged(decimal price) { ... } ... } public delegate void Notify(decimal newPrice); Stock obs = new Stock(); Notify notify = new Notify(obs.ProcessPriceChanged); notify(123); // corre o método guardado </pre>
	<p>Em C#, existe o tipo delegate cujas variáveis guardam referências de métodos. A definição destas variáveis usa a assinatura do método que guarda. O nome da variável é fornecido no lugar do nome do método usado na assinatura. No exemplo acima as variáveis do tipo delegate com o nome Notify podem guardar métodos que não tenham tipo de retorno e que recebam um valor do tipo decimal como argumento.</p> <p>Um delegate pode guardar mais do que um método, podendo ser acrescentados métodos usando o operador += e retirados usando o operador -=.</p> <p>Os métodos guardados nos <i>delegates</i> podem retornar um valor mas, neste caso, apenas o valor retornado pelo último método que foi guardado no <i>delegate</i> é devolvido. Também no caso de um dos métodos gerar uma exceção, esta exceção propaga-se aos outros métodos não sendo executado nenhum dos métodos seguintes.</p>	
	JAVA	C#
		<pre> public class MyEventNotifier { public event Notify NotifyCallBack; } </pre>
	<p>Para simplificar o uso de <i>delegates</i> o C# introduz o tipo event que cria um delegate na classe implicitamente. Neste caso ao expor-se o event permite-se uma inscrição fácil dos métodos a serem considerados através da utilização dos operadores += e -=.</p> <p>A utilização de eventos, que devem ser públicos, evita que se tenham de criar variáveis do tipo delegate públicas.</p>	